VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR ROBOTICKÉHO PRACOVIŠTĚ

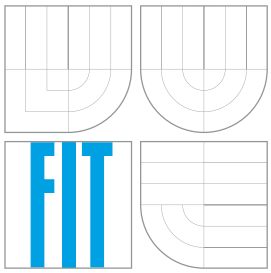
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN HRABOŠ

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR ROBOTICKÉHO PRACOVIŠTĚ

SIMULATOR OF ROBOTIC ARM WORKCELL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN HRABOŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADIM LUŽA
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace pro simulaci robotického ramene. Jedná se o skutečný model robota, který je umístněný ve Fakulte informačních technologií v Brně. Robotem je možné libovolně manipulovat a taky přepínat pohled na scénu ze šesti kamer.

Abstract

This thesis describes design and implementation of an application for simulating robotic armcell. The real model of this robot is located in the Faculty of Information Technology in Brno. It is possible to control the moves of the robotic arm and also switch the view at the scene from six cameras.

Klíčová slova

Simulace, robotika, model, grafické rozhraní, OpenRAVE, Qt.

Keywords

Simalation, robotics, model, graphical interface, OpenRAVE, Qt.

Citace

Martin Hraboš: Simulátor robotického pracoviště, bakalářská práce, Brno, FIT VUT v Brně, 2013

Simulátor robotického pracoviště

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radima Luži. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Hraboš
14. května 2013

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu Ing. Radimu Lužovi za vedění a pomoc při psaní této práce.

Obsah

Kapitola 1

Úvod

V dnešnej dobe je čoraz častejšie počuť o robotoch a robotike. V mnohých prípadoch si to ani neuvedomujeme, ale roboti nás sprevádzajú každý deň a uľahčujú nám prácu, ktorú by sme museli vykonávať manuálne alebo by sme ju nevykonali vôbec, pretože by sme toho skrátka neboli schopní. Existujú však roboti, s ktorými bežný človek do styku neprichádza, respektíve neexistujú prostriedky na to, aby s nimi mohol do styku prísť. Z tohto dôvodu existujú rôzne simulátory, ktoré sa snažia robota človeku čo najviac priblížiť.

Táto bakalárska práca popisuje návrh, implementáciu a testovanie simulátoru pre robotické pracovisko. Konkrétne sa jedná o robotické rameno, ktoré je umiestnené v areáli fakulty v miestosti A223.1. Cieľom tejto práce je zamerať sa na vytvorenie jednoduchého intuitívneho grafického užívateľského rozhrania, pomocou ktorého je možné simulovať všetky pohyby robotického ramena a zobraziť ich na obrazovku. Pohľad na rameno je možné prepínať, pričom každý z možných pohľadov odpovedá pohľadu jednej zo skutočných kamier, ktorá sníma robotické rameno. Celú prácu je možné rozdeliť do 3 logických častí:

Prvá časť zdôrazňuje už známe fakty, ktoré nepotrebujú opodstatnenie. Popisuje známe prostriedky a princípy, ktoré je možné uplatniť pri realizácii a analyzuje výhody a nevýhody každého z nich. Pre porovnanie sú tu uvedené už vytvorené, voľne dostupné simulačné aplikácie vytvorené inými ľuďmi, z ktorých je možné čerpať inšpiráciu.

Druhá časť sa zaoberá procesom dekompozície problémov, návrhu a realizácie samotnej aplikácie. Tu je uvedené, akým smerom sa bude aplikácia uberať a ktoré prostriedky som sa rozhodol použiť k realizácii a prečo. Je tu popísaná samotná implementácia, ktorá tvorí asi najdôležitejší prvok celého diela. Sú tu uvedené vlastné postupy a myšlienky, ktoré som použil pri tvorbe aplikácie. Ďalej sú tu zhrnuté všetky informácie, problémy a zaujímavosti, na ktoré som pri implementácii narazil.

Posledná časť zahrňuje kapitoly testovanie a záver. Zhrňuje teda údaje, ktoré boli získané testovaním aplikácie na náhodných ľuďoch. Ďalej uvádza prípadné nedostatky a rozšírenia, ktoré by bolo vhodné implementovať v ďalších verziách aplikácie. Záver sumarizuje všetko podstatné o celej práce. Analyzuje čo sa mi podarilo, ale aj nepodarilo, prípadne veci, ktoré by sa dali urobiť inak alebo efektívnejšie.

Kapitola 2

Teória

Neodmysliteľnou súčasťou každého diela je teoretický základ, na ktorom je celá práca postavená a z ktorého čerpá všetky podstatné informácie. Síce zmyslom tejto práce je skôr experimentovanie a použitie vlastných ideí a praktík, zopár teoretických faktov je predsa len potrebné zdôrazniť.

2.1 Simulácia

Pod pojmom simulácia v našom prípade rozumieme počítačový program, ktorý sa snaží simulovať model daného systému [1]. Simuláciu reálneho robota je vhodné využiť najmä vtedy, keď nepotrebujume fyzicky manipulovať so strojom ale iba zistiť, prípadne prezentovať jeho možnosti a schopnosti. Preto je potrebné, aby simulačný nástroj čo najviac odpovedal realite. Z tohto dôvodu musia všetky pohyby modelu na obrazovke korešpondovať so skutočnými pohybmi ramena. Takýto prostriedok je potom možné verejne prezentovať a zaujať ním cieľených užívateľov.

Simulácia je výskumná metóda, ktorej podstata spočíva v tom, že skúmaný dynamický systém nahradíme jeho simulátorom a s ním realizujeme pokusy s cieľom získať informácie o pôvodnom skúmanom systéme [9]. Simuláciu systémov je možné uskutočňovať mimo reálne objekty, bez ovplyvnenia skutočnej prevádzky resp. bez existencie reálneho, skúmaného systému. Výsledkom simulácie sú informácie o skúmanom systéme a jeho prvkoch podľa definovaných parametrov. Informácie o skúmaných prvkoch, objektoch systému je možné získať:

- experimentovaním s reálnym objektom,
- experimentovaním s modelom,
- riešením modelu analytickými alebo numerickými metódami.

2.1.1 Dynamický systém

Pri simulácii systému nás najviac zaujíma chovanie systému v závislosti na čase. Systém, ktorého príznaky sa menia v čase sa nazýva *dynamický systém* [8]. *Čas* je nezávislá premenná, ktorá určuje vzájomnú odľahlosť medzi minulými a budúcimi udalosťami systému. *Čas systému* považujeme za reálnu nezávislú premennú, spojitú veličinu, ktorá určuje odľahlosť medzi minulými a budúcimi udalosťami systému. Ak je udalosť U_2 vykonaná neskoršie

ako U_1, potom pre čas platí $t_2 > t_1$. Závislosť udalostí je taká, že z minulej udalosti vyplýva budúca udalosť.

Základnou vlastnosťou dynamického systému je, že jeho chovanie v ktoromkoľvek časovom okamihu nezávisí iba na príznakoch, ktoré naň pôsobia práve v tomto okamihu, ale taktiež na príznakoch, ktoré naň pôsobili v minulosti. Systém pracuje tak, akoby mal pamäť na predchádzajúce príznaky.

2.1.2 Klasifikácia simulácie

Simuláciu možno rozdeliť podľa charakteru zmeny stavu modelu v čase na:

- **diskrétnu** – stav systému sa skúma v diskrétnych časových intervaloch,
- **spojitú** – stav systému sa skúma spojito,
- **kombinovanú.**

Podľa typu modelu rozdeľujeme simuláciu na:

- **deterministickú** – výsledky sú stanovené stavom v časovom okamihu t_0,
- **stochastickú** – výsledky sú stanovené s určitou pravdepodobnosťou.

2.1.3 Simulačné jazyky

Všeobecné programovacie jazyky sa ukázali ako ťažkopádne a málo dynamické na vytváranie rýchlych zmien v simulačných modeloch. Simulačné jazyky sú prispôsobené pre potreby simulácie, umožňujú tvorbu programových simulačných modelov v takej podobe, aby vytvorené modely čo najlepšie odzrkadľovali simulované systémy.

Simulačné jazyky delíme na jazyky orientované na:

- aktivity,
- udalosti,
- procesy.

Delenie simulačných jazykov podľa spôsobu simulácie:

- **Spojitá simulácia** – Simula, Java,
- **Diskrétna simulácia** – SLAM, DYNAMO,
- **Kombinovaná simulácia** – Scilab, Simulink, Modelica, Dymola.

2.2 Popis dynamiky robota

Dynamiku robota je možné popísať diferenciálnymi rovnicami, ktoré sa dajú jednoducho odvodiť z métod analytickej mechaniky.

1. **Lagrangeové rovnice.** Vychádzame z Lagrangeových rovníc 2. rádu, ktoré majú tvar

$$\frac{\partial}{\partial t}.\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = Q_j, \quad j = 1, \ldots, n \tag{2.1}$$

kde $L = K - V$ je tzv. Lagrangeová funkcia a Q_j sú tzv. zovšeobecnené sily, ktoré neboli zahrnuté do potenciálnej energie V, potom môžme celkovú kinematickú energiu vyjadriť vzťahom

$$K = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}} \tag{2.2}$$

Do rovníc môžme zahrnúť aj rovnice pohonov, prípadne prevody a vplyvy trecích síl či tlmenia. Po zložitejších úpravach pohybovú rovnicu robota vyjadríme v maticovom tvare

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \mathbf{u} \tag{2.3}$$

kde

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \left[\frac{1}{2}\dot{\mathbf{M}}(\mathbf{q}) + \mathbf{S}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{M_0}\right] \tag{2.4}$$

V uvedených vzťahoch vektor

$$\mathbf{q} = (\mathbf{q_1}, \ldots, \mathbf{q_n})^{\mathbf{T}} \tag{2.5}$$

predstavuje **zovšeobecnené súradnice**, ktoré sú vzájomne nezávislé. Znak V tu reprezentuje potenciálnu energiu. $M(q)$ je symetrická pozitívne definičná matica, spojitá vzhľadom ku všetkým svojím parametrom. M_0 predstavuje maticu reprezentujúcu vplyvy trenia a tlmenia a S je antisymetrická matica tvaru

$$S_{i,j}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\left[\sum_{k=1}^{n} \dot{q}_k \frac{\partial M_{i,k}}{\partial q_j} - \sum_{k=1}^{n} \dot{q}_k \frac{\partial M_{j,k}}{\partial q_i}\right] \tag{2.6}$$

Veličina **u** predstavuje vektor riadenia. Môže byť reprezentovaný vstupnými krútiacimi momentami, alebo vstupnými riadiacimi elektrickými prúdmi.

2. **Hamiltonove rovnice.** Pokiaľ použijeme Hamiltonove kanonické rovnice, odvodíme analogickým postupom vzťahy popisujúce dynamické chovanie robota v nasledujúcom tvare:

$$\dot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\mathbf{p} \tag{2.7}$$

$$\dot{\mathbf{p}} = \frac{1}{2}\dot{\mathbf{M}}\dot{\mathbf{q}} - \mathbf{S}\mathbf{M}^{-1}\mathbf{p} - \mathbf{q}(\mathbf{q}) + \mathbf{F} \tag{2.8}$$

Vektor **p** predstavuje tzv. zovšeobecnenú hybnosť, vektor **F** reprezentuje vektor zovšeobecnených vonkajších síl (obvykle krútiace momenty) a môže byť priamo vektorom riadenia. Obe uvedené vektorové rovnice tak predstavujú sústavu 2n diferenciálnych rovníc prvého rádu [10].

2.3 Užívateľské rozhranie

Užívateľské rozhranie je jediný prostriedok, ktorý má užívateľ k dispozícii na ovládanie simulácie. Preto musí byť čo najjednoduchšie a zároveň musí poskytovať všetky požadované funkcie. V neposlednom rade musí mať z rozhrania pozitívny dojem aj obyčajný laik, ktorý nikdy pred tým nebol v kontakte s podobným nástrojom a nezaoberá sa jeho problematikou. Ak by užívateľ z aplikácie alebo z jeho hociktorého prvku nemal dobrý dojem, automaticky odmietne celú aplikáciu a začne hľadať inú, užívateľsky prívetivejšiu alternatívu. Preto netreba návrh rozhrania podceniť a treba si jasne ujasniť ciele. V ideálnom prípade je vhodné urobiť prieskum medzi ľuďmi a zo získaných výsledkov vyvodiť adekvátne závery a následne podľa ich implementovať aplikáciu. V našom prípade sa treba zamerať na cieľovú skupinu užívateľov, teda na tých, ktorí budú aplikáciu reálne využívať. Je nezmyselné robiť prieskum medzi ľuďmi, ktorí tejto problematike vôbec nerozumejú.

2.4 Dostupné nástroje

2.4.1 Jazyk C++

Tento jazyk vytvoril v roku 1983 Bjarne Stroustrup. Neskôr prevzala zodpovednosť za jeho vývoj firma *Bell Labs.* Môže sa na prvý pohľad zdať, že už nie je konkurenciou pre novšie programovacie jazyky, ale opak je pravdou. Pokiaľ napríklad chceme napísať program, ktorý má byť bez väčších problémov presnositeľný medzi rôznymi platformami, je voľba tohoto jazyku celkom logická [7].

Z týchto dôvodov je vhodné použiť tento programovací jazyk aj pri tvorbe našej aplikácie, pri ktorej sa kladie dôraz na prenositeľnosť. Ďalšou výhodou je možnosť využiť objektovo orientované programovanie, ktoré nám uľahčí prácu pri vkladaní grafických ovládacích prvkov. Objekty sú takisto nutné na prepojenie s knižnicou Qt4. Jazyk C++ je možné prekladať do binárneho kódu napríklad pomocou prekladača *gcc,* resp. *g++* v prostredí operačného systému *Unix* alebo pomocou prekladačov *MinGW* alebo *MSVC* v operačnom systéme *MS Windows.*

2.4.2 Qt4

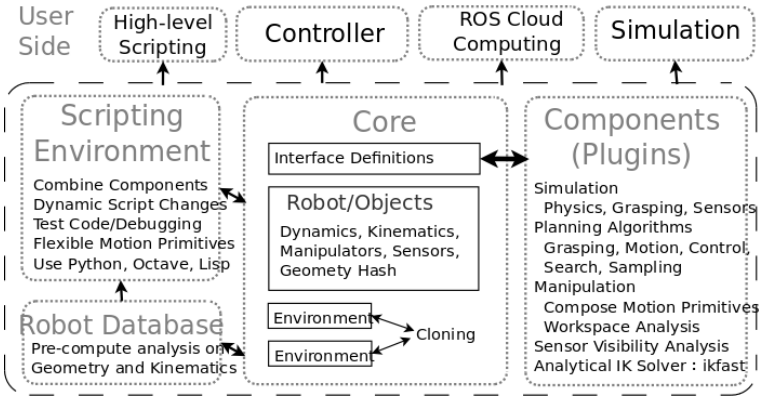
Qt je komplexná C++ aplikačná vývojová štruktúra na vytváranie multi-platformných GUI aplikácii, ktorá využíva „napíš raz, prekladaj kdekoľvek“ koncepciu. Qt umožňuje programátorom využívať jeden zdrojový strom pre aplikácie spustiteľné na Windows 98 až Windows 7, Mac OS X, Linux, Solaris, HP-UX a mnohých iných verziách Unix-u s X11. Qt knižnice a nástroje sú okrem iného súčasťou Qt/Embedded Linux, produkt, ktorý poskytuje svoj vlastný systém okien na vrchnej vrstve vstavaných systémoch Linux [6].

Túto knižnicu je vhodné použiť na vytvorenie grafického užívateľského rozhrania, pretože umožňuje celkom jednoduchú manipuláciu prvkov pri využití objektovej koncepcie jazyka C++. Jeho výhodou je opäť prenositeľnosť a rozšírenosť medzi programátormi. Súčasťou knižnice Qt4 sú aj ďalšie pomocné nástroje, ktoré uľahčujú tvorbu aplikácie. Pri akomkoľvek probléme nie je problém vyhľadať potrebné informácie na riešenie problému na internete.

2.4.3 OpenRAVE

OpenRAVE poskytuje prostredie pre testovanie, vývoj a návrh algoritmov pre robotické aplikácie v reálnom svete. Hlavný dôraz sa kladie na simuláciu a analýzu kinematickej a geometrickej informácie vo vzťahu k pohybovým algoritmom. Myšlienka jednotnosti OpenRAVE umožňuje, že táto technológia môže byť jednoducho integrovaná do robotických systémov. Dostupných je mnoho nástrojov pre príkazový riadok, ktoré pracujú s robotmi a plánovačmi, a jadro tohto nástroja je dostatočne malé, aby mohlo byť použité vo vnútri ovládačov, ale aj vo väčších zariadeniach. Dôležitým cieľom tohto nástroja je automatizácia priemyselnej robotiky [5].

V našej aplikácii je vhodné použiť OpenRAVE, pretože je napísaný v jazyku C++ a ponúka nám všetky potrebné prostriedky na realizáciu grafickej simulácie. Samotné grafické rozhranie je vytvorené pomocou Qt4, preto je možná integrácia tohto nástroja do hociakej aplikácie využívajúcej Qt4. Definícia modelu je zabezpečená pomocou textového súboru vo formáte *XML*. OpenRAVE umožňuje definovanie modelu aj pomocou iných formátov, napr. *.zae, .dae.*



Obr. 2.1: Štruktúra OpenRAVE

2.4.4 Qt Creator

Qt Creator je multi-platformné integrované vývojové prostredie pre vývoj aplikácií v jazyku C++ s využitím Qt knižnice na tvorbu grafického užívateľského rozhrania. Tvorba aplikácií pomocou tohto nástroja je výhodná v tom, že dokážeme odtieniť zdrojový kód aplikácie od kódu grafického rozhrania. Na návrh rozhrania je prípadne možné použiť aj pomocné nástroje Qt Designer a Qt Quick Designer. Nástroj *Qt Designer* slúži na jednoduchú interaktívnu tvorbu grafických prvkov, ktoré sú uložené do *XML* súboru. Nástroj *Qt Linguist* umožňuje jednoducho vytvoriť preklad aplikácie do ľubovoľného národného jazyka. Pretože je Qt Creator taktiež multi-platformný, nebráni nám nič v tom, aby sme mohli písať zdrojové kódy na rôznych platformách a tak postupne aj otestovali nami vytvorenú aplikáciu na rôznych systémoch.

Na začiatku tvorby každej Qt aplikácie je potrebné vytvoriť pomocou nástroja *qmake* nový projekt. Potom môžme začať písať našu aplikáciu, kde na začiatku musíme vytvoriť inštanciu triedy *QApplication*.

Kapitola 3

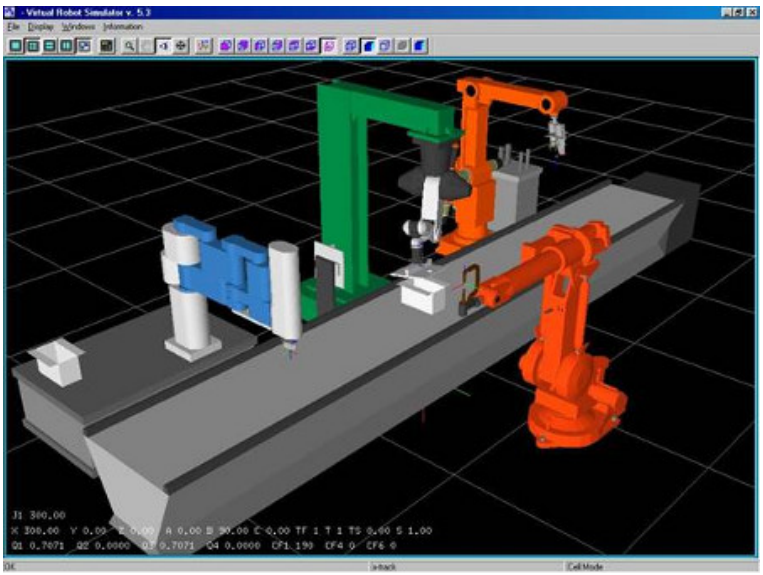
Dostupné simulátory

Počítačové simulátory máme možnosť vidieť na každom kroku. Môžme s nimi prísť do kontaktu v každom priemyselnom odvetví a takisto pri využívaní rôznych služieb alebo pri vzdelávaní a výskume. Príkladom môžu byť trenažéry v autoškole, letecké simulátory, simulátory vyťaženia počítačových sietí a rôzne iné. Simuláciou taktiež ušetríme nemalé finančné prostriedky, ako keby sme si mali zadovážiť reálny nástroj.

3.1 VirtualRobot Simulator

VirtualRobot je voľne dostupný softvér, ktorý v sebe obsahuje viac programov vytvorených v jazyku C++ pre robotické aplikácie, výskum a vzdelávanie s grafickou reprezentáciou založenou na OpenGL na operačných systémoch Microsoft Windows (2k/NT/9x). Jeho začiatok, v roku 1998, bol motivovaný potrebou vzdialených grafických aplikácií pre programovanie, simulovanie a monitorovanie multi-robotických jednotiek pod kontrolou systémom GENERIS (Generalised Software Control System for Industrial Robots).

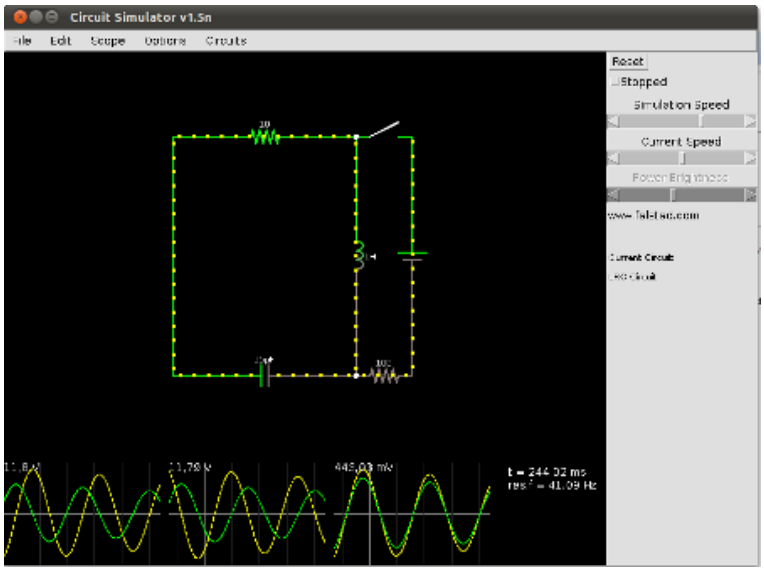
VirtualRobot Simulator je platforma na simuláciu, ktorá obsahuje dynamicky linkované knižnice a externé aplikácie (VRS Tools, VRM Tools a VRS Demos) [2].



Obr. 3.1: VirtualRobot Simulator

3.2 Circuit simulator

Jedná sa o applet pre webové prehliadače napísaný v jazyku Java, ktorý slúži na simuláciu elektrických obvodov. Jeho výhoda spočíva v tom, že nemusíme ručne počítať zložité rovnice, ale tento simulátor nám hneď zobrazí hodnoty všetkých potrebných elektrických veličín. Taktiež nám zobrazí toky elektrického prúdu, ktoré v reálnom živote nemáme možnosť sledovať. Súčasťou appletu je aj mnoho vopred nadefinovaných príkladov, na ktorých sa užívateľ naučí používať tento nástroj. Samozrejmosťou je vytváranie a editovanie vlastných elektrických obvodov s možnosťou uloženia na disk [3].

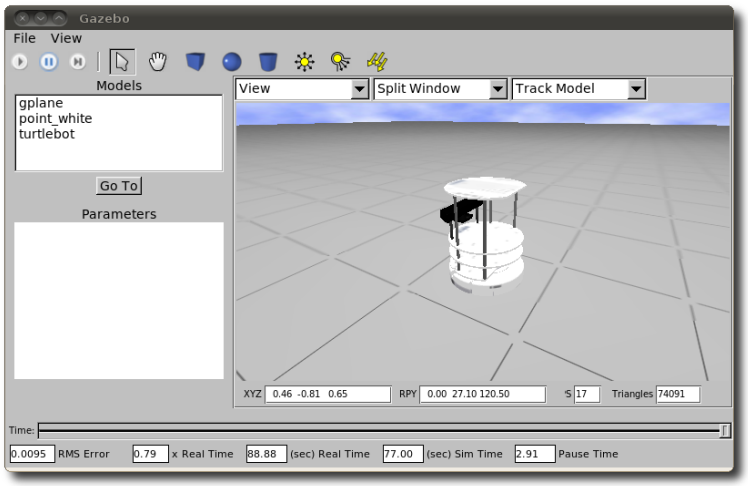


Obr. 3.2: Circuit simulator

3.3 Gazebo

Gazebo je multi-robotický simulátor Je schopný simulovať rôznych robotov, senzory a objekty v trojrozmernom svete. Generuje skutočnú spätnú väzbu senzorov ale aj fyzické interakcie medzi objektami. Gazebo bol pôvodne navrhnutý ako pomocný nástroj pri vývojovom procese algoritmov pre robotické systémy. Mnoho vývojárov využívalo Gazebo taktiež na vývoj a vykonávanie experimentov v simulovanom prostredí. Gazebo je integrovaný vo vývojom prostredí ROS (Robot Operating System).

Svoju existenciu začal v roku 2002 na americkej univerzite *University of Southern Carolina*. Pôvodní zakladatelia boli Dr. Andrew Howard a jeho študent Nate Koenig. V roku 2009 John Hsu integroval *ROS* a *PR2* do Gazeba, ktoré sa odvtedy stalo primárnym nástrojom používaným v komunite ROS. V roku 2011 za finančnej podpory napredovalo a v blízkej budúcnosti sa očakáva dokončenie nástroja, ktorý bude užívateľsky prívetivý, stabilný, multiplatformný a pripravený na tvorbu dizajnu pre robotov a robotické aplikácie [4].



Obr. 3.3: Gazebo

Kapitola 4

Návrh aplikácie

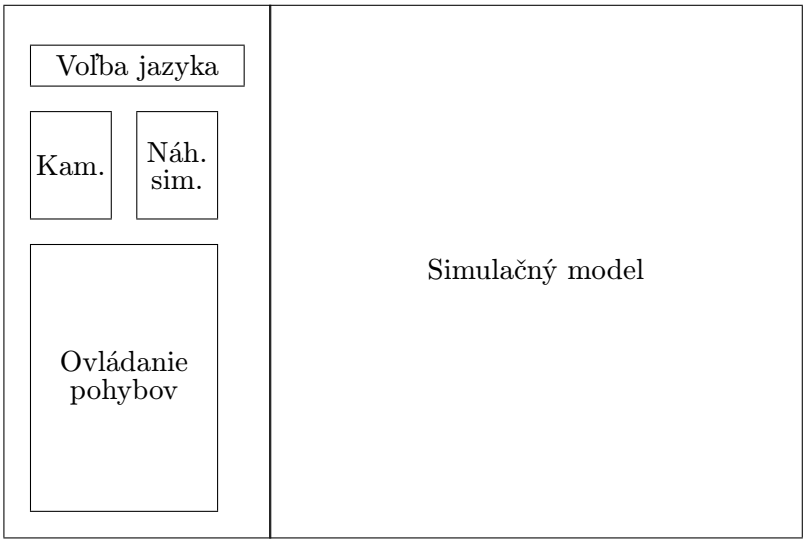
Návrh samotnej aplikácie zohráva dôležitú úlohu vo vývojovom cykle každého softvéru. Preto je potrebné premyslieť každý detail, aby sme na jednej strane dokázali zákazníka našou aplikáciou uspokojiť a na druhej strane, aby naša aplikácia vôbec bola realizovateľná. Pri návrhu je dobré zistiť a zohľadniť požiadavky, ktoré ľudia od výslednej aplikácie požadujú. Návrhu našej aplikácie v prvom rade predchádza oboznámenie sa so skutočným robotickým ramenom a s pozíciou kamier, ktoré sú spolu s robotom umiestnené v jednej miestnosti.

4.1 Schéma rozhrania

Našu aplikáciu možno rozčleniť na viaceré logické celky:

- Užívateľské rozhranie
  - Ovládanie pohybov,
  - Ovládanie kamier,
  - Voľba jazyka,
  - Náhoda simulácia.
- Simulačný model
  - Statické časti robota,
  - Dynamické časti robota,
  - Kĺby,
  - Senzory.

Obrázok 4.1 znázorňuje prototyp aplikácie s uvedením pozície ovládacích prvkov a simulačného modelu.



Obr. 4.1: Schéma rozhrania

4.2 Užívateľské rozhranie

4.2.1 Ovládanie pohybov

Táto časť obsahuje jednoduché tlačidlá na ovládanie pohybov robotického ramena. Každá časť robota je označená obrázkom a je ju možné ovládať samostatne dvoma smermi. Smer pohybu je reprezentovaný dvomi tlačidlami s označením + a -. Pre každú pohyblivú časť robota sú teda potrebné dve tlačidlá.

4.2.2 Ovládanie kamier

Na ovládanie kamier je potrebných šesť tlačidiel. Na jednú kameru pripadá jedno tlačidlo. Po stlačení tohto tlačidla sa simulačný model preklesí z takého pohľadu, z akého by príslušná kamera snímala skutočného robota. V prípade týchto tlačidiel je vhodné vytvoriť ich inštanciou triedy *QPushButton* a následne nastaviť *setCheckable(true)*, čím zabezpečíme, aby bolo vždy vybraté iba jedno tlačidlo. Týmto spôsobom budeme mať dostupnú informáciu o tom, ktorá kamera je práve aktívna.

4.2.3 Voľba jazyka

Aby sme nezohľadnili len niektorých užívateľov aplikácie, je možné nastaviť jazyk, ktorým s nami program komunikuje. V našich zemepisných šírkach je vhodné použiť predovšetkým anglický jazyk, ktorým zabezpečíme univerzálnosť použitia. Tomuto jazyku zvyčajne rozumejú všetci užívatelia pohybujúci sa v technologických sférach. Okrem anglického jazyka je vhodné implementovať ešte český a slovenský jazyk. Zmenu požadovného jazyka je možné vykonať kliknutím na príslušné tlačidlá.

4.2.4 Náhodná simulácia

Táto časť predstavuje možnosť spustenia automatickej náhodnej simulácie, ktorá demonštruje všetky možné pohyby robota. Je taktiež možné sledovať pohyby zo všetkých dostupných ka-

mier. Ovládacie prvky robota je však pri vykonávaní náhodnej simulácii nutné zablokovať, aby užívateľa nelákalo ovládať robota počas bežiacej simulácie.

4.3 Simulačný model

Simulačný model predstavuje približne polovicu plochy, ktorá ja určená pre aplikáciu. Je lokalizovaný v pravej časti zobrazovaného okna. Na tejto ploche sa prekresľuje model podľa toho, aké príkazy mu užívateľ dáva. Celý model je vytvorený pomocou nástroja OpenRAVE, ktorý umožňuje jeho ovládanie a pomerne jednoduchú manipuláciu s ním.

Kapitola 5

Implementácia

5.1 Užívateľské rozhranie

Užívateľské rozhranie je vytvorené pomocou knižnice Qt4, konkrétne inštanciou triedy `QMainWindow`. Pomocou tejto triedy je vytvorené hlavné okno aplikácie, ktoré využíva záhlavie generované grafickým manažérom používaného operačného systému. Rozhranie možno rozdeliť na tieto časti:

1. **Voľba jazyka** – pozostáva z 3 tlačidiel vytvorených inštanciou triedy `QPushButton`. Tieto tlačidlá obsahujú ikonu s vlajkou príslušnej krajiny a kód jazyka používaného v danej krajine[1]. Tlačidlá sú vložené do skupiny *QGroupBox,* čiže tvoria jednu skupinu tlačidiel a majú nastavenú vlastnosť *autoExclusive,* čo zaručuje, že môže byť vybraté iba jedno tlačidlo[2].

2. **Pohľad z kamery** – obsahuje 4 tlačidlá, ktoré reprezentujú skutočné kamery. Princíp fungovania tlačidiel je rovnaký ako pri voľbe jazyka (1).

3. **Ovládanie pohybov** – tvorí 12 tlačidiel a 6 obrázkov, z ktorých každý obrázok reprezentuje práve jednu časť simulačného modelu, s ktorou je možné pohybovať. Ku každému obrázku prislúchajú dve tlačidlá (označené + a -), ktoré ovládajú smer pohybu. Obrázky sú vložené pomocou objektu `QLabel,` ktorý obsahuje pixmapu `QPixmap.` Veľkosť obrázka je ešte upravená pomocou metódy `QPixmap::scaledToHeight(50).` Obrázky sú uložené v súbore *Resources.qrc,* pôvodne vo formáte *png.*

4. **Náhodná simulácia** – pozostáva z dvoch tlačidiel – *štart* a *stop.* Po kliknutí na *štart* je spustená náhodná simulácia, ktorá je aktívna až do ukončenia simulácie kliknutím na tlačidlo *stop.* Spustením náhodnej simulácie sa deaktivuje ovládanie pohybov, aby sa zamedzilo prípadným konfliktom pri simulácii.

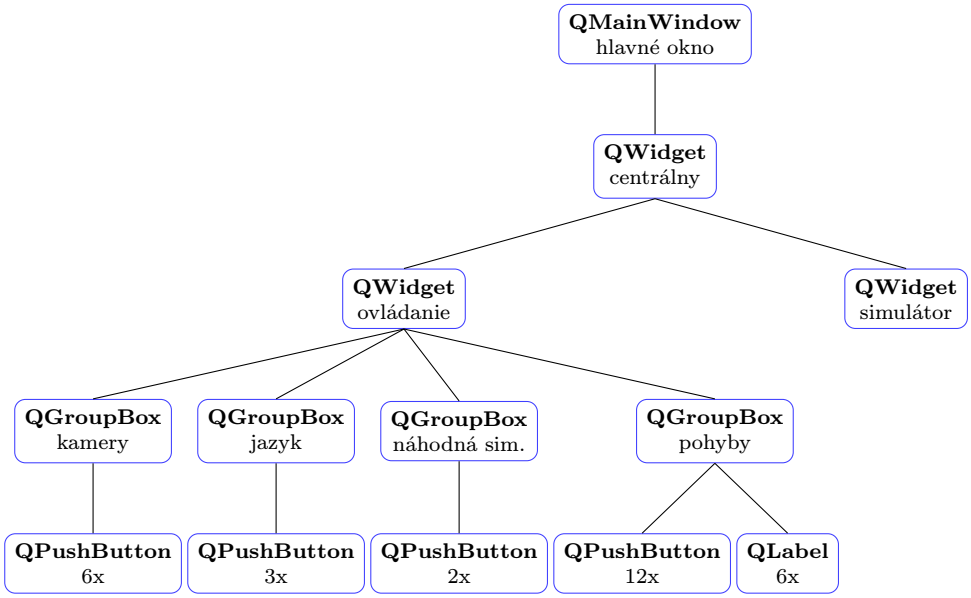
5. **Model OpenRAVE** – tvorí pravú časť okna. Ovládacie prvky, ktoré sú vytvorené v rozhraní OpenRAVE sú zablokované *(Bližší popis je v sekcii 5.2).*

[1] Kód jazyka je podľa normy ISO 639-1. http://www.sochorek.cz/archiv/slovniky/iso-639/s.htm
[2] Ekvivalentné s *Radio button*

5.1.1 Tvorba ovládacích prvkov

Všetky grafické prvky sú vytvorené nástrojom *Qt 4 Designer,* ktorý je integrovaný v aplikácii *Qt Creator.* Grafické rozhranie nie je teda priamo definované v zdrojovom kóde[3], ale automaticky generované aplikáciou do súboru `mainwindow.ui`. Tento súbor je textového formátu *XML* a pri preklade je najskôr preložený do jazyka C++ (vznikne súbor `ui_mainwindow.h`).
Celkovo sú v aplikácii použité tieto grafické prvky:

- 23-krát QPushButton – ovládanie kamier, pohybov, náhodnej simulácie a voľby jazyka,
- 6-krát QLabel – obrázky reprezentujúce ovládateľné časti robota,
- 4-krát QGroupBox – zapúzdrenie tlačidiel do jedného celku (jazyk, náhodná simulácia a pohyby).
- 3-krát QWidget – jeden centrálny, ktorý zapúzdruje ďalšie dva,
- 1-krát QMainWindow – hlavné okno aplikácie.



Obr. 5.1: Schéma ovládacích prvkov rozhrania (Qt4)

5.1.2 Obrázky a ikony

Obrázky sú vložené pri nasledujúcich prvkoch:

- **Voľba jazyka** – tlačidlá sú doplnené o ikony – vlajky, ktoré reprezentujú danú krajinu. Ikony sú vložené priamo v nástroji *Qt 4 Designer.*

[3] Výnimku tvorí *QWidget* rozhrania OpenRAVE, ktorý je pridaný dodatočne.

- **Pohyby robota** – obrázky (časti robota) sú vytvorené pomocou triedy `QPixmap` a vložené do objektu `QLabel` volaním metódy `QLabel::setPixmap(QPixmap)`. Samotné vkladanie obrázkov je implementované v metóde `QMainWindow::setPictures()`.

Všetky obrázky použité v aplikácii sú uložené v tzv. *resources*. Tým sa vyhneme prípadným konfliktom so súborovým systémom pri zadávaní globálnej alebo lokálnej cesty. Všetky „resources“ sú uložené v súbore `Resources.qrc` v adresári s binárnym súborom.

5.2 Integrácia prostredia OpenRAVE

Simulačný nástroj OpenRAVE tvorí samostatnú aplikáciu s vlastným užívateľským rozhraním, ktoré je možné integrovať do inej aplikácie, ktorá používa pre vykreslenie GUI knižnicu Qt4[4]. Integrácia je možná dvomi spôsobmi:

- Celá aplikácia pozostáva z jedného okna, v ktorom je integrované okno z OpenRAVE,
- Aplikácia pozostáva z dvoch okien, kde jedno je grafické rozhranie OpenRAVE a druhé vytvorené grafické rozhranie.

V našej aplikácii je využitá možnosť integrácie do jedného okna, ktoré tvorí komplexnú aplikáciu. Rozhranie OpenRAVE je vytvorené volaním metódy `OpenRAVE::RaveCreateViewer(penv, "qtcoin")`[5] Integrácia je prevedená funkciou `Dynamic_cast<QWidget*>`, ktorá vytvorí z ukazateľa na okno ukazateľ na `QWidget`, ktorý už nie je problém pridať do vytváraného rozhrania: `ui->layout->addWidget(QWidget*)`. Keďže je okno umiestnené v *layout-e*, prisbôsobi sa zmenám veľkosti okna. Nastavením `viewer->setDisabled(true)` zablokujeme všetky ovládacie prvky z prostredia OpenRAVE, okrem možnosti priblíženia, vzdialenia a posunu myšou.

5.3 Kamery

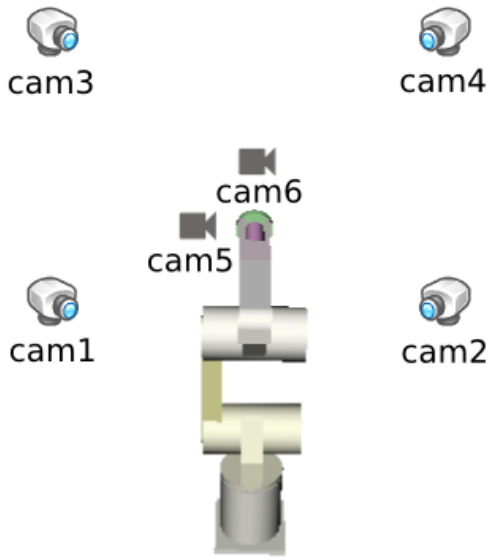
Kamery snímajúce robota možno rozdeliť do dvoch kategórií:

1. Kamery pevne umiestnené na stene po stranách robotického ramena.
2. Kamery umiestnené priamo na ramene robota (ich pozícia sa mení s pohybom robota).

Všetky kamery vlastne iba reprezentujú pohľad na robotické rameno. Na obrázku 5.2 je znázornené rozmiestnenie kamier v areáli fakulty. Kamery č. 1 až 4 majú pevne danú pozíciu. Pri kamerách č. 5 a 6 už nestačí iba pohľad na scénu, ale je nutná implementácia senzorov v definícii modelu (viď definícia kamery č. 5 v A). Natáčenie samotných kamier je v aplikácii zanedbané. Pri nutnej potrebe tejto funkčnosti je možné využiť rozhranie OpenRAVE, ktoré dokáže zobraziť scénu z hociakého pohľadu.

[4] Verzia OpenRAVE 0.9.0 je nekompatibilná s aktuálnou verziou Qt5.

[5] Parameter *penv* – ukazateľ na protredie, parameter *"qtcoin"* – typ zobrazovaného prostredia OpenRAVE.



Obr. 5.2: Pozícia kamier v učebni A223.1 v areáli FIT v Brne

5.3.1 Kamery č. 1 až 4

Pri implementácii pohľadu týchto kamier je využitá trieda `Transform` z knižnice OpenRAVE, kde sú nastavené konštantné hodnoty pohľadu na scénu. Metódou `setCamera()` sú nakoniec nastavné zadané hodnoty a scéna je prekreslená z požadovaného uhlu.

Príklad implementácie pohľadu kamery č. 1:

```
RaveTransform pose;
pose.trans.x = -0.775708;
pose.trans.y = 1.16778;
pose.trans.z = 1.33003;
pose.trans.w = 0;

pose.rot.x = 0.234403;
pose.rot.y = 0.945392;
pose.rot.w = 0.21156;
pose.rot.z = -0.0808181;

viewer->SetCamera(pose);
```

5.3.2 Kamery č. 5 a 6

Tieto kamery fungujú ako senzory umiestnené na robotickom ramene. Sú načítané z XML súboru, ktorý definuje model. To, že sa jedná o kamery je nastavené atribútom

`type="BaseCamera"` v značke `<sensor>`. Pri spustení aplikácie sú obe inicializované a nastavené ako vypnuté volaním metódy `Configure(SensorBase::CC_PowerOff)`. Inicializácia týchto kamier je obsiahnutá v metóde `setSensors()`. Po kliknutí na danú kameru sa aktivuje senzor volaním `Configure(SensorBase::CC_PowerOn)`. Následne je získaná pozícia kamery v súradnicovom systéme a nastavený pohľad na scénu metódou `setCamera()`. Po nastavení pozície je daný senzor opäť vypnutý.

V prípade náhodnej simulácie alebo pri viacnásobnom pohybe[6] z pohľadu týchto kamier je nutné nastaviť pozíciu po každom jednom kroku simulácie pohybu. V každom kroku dochádza teda k aktivácii senzoru, získaniu pozície kamery, nastaveniu pohľadu a deaktivácii senzoru. Otázne je prečo je teda nutné aktivovať a následne deaktivovať senzor v každom kroku. Týmto činom sa však vyhneme mrznutiu aplikácie, ku ktorému dochádza pri spustenej simulácii v kombinácii s aktivovanými senzormi.

5.4 Náhodná simulácia

Náhodná simulácia predstavuje taký režim behu aplikácie, pri ktorom užívateľ nijak nekoriguje pohyby robota. Robot vykonáva náhodne generované pohyby v pevne daných časových intervaloch. Tento režim simulácie prebieha v nekonečnej slučke, preto je nutné zabezpečiť, aby prislúchajúce výpočty prebiehali vo vlastnom vlákne alebo procese a nezaťažovali tak vlákno rozhrania. Pri implementácii tohto režimu je využitá knižnica `Boost` a je vytvorené nové vlákno pomocou triedy `boost::thread`. Náhodnú simuláciu reprezentuje metóda `MainWindow::run()`, ktorá je spúšťaná v tomto vlákne. Generovanie náhodných pohybov je zabezpečené metódou `RaveRandomInt()`, ktorá generuje náhodné celé čísla. Tieto čísla sú následne upravené operáciou *modulo,* čím zabezpečíme potrebný rozsah čísel. Rozsah čísel získame pomocou metódy `GetDOF()`, ktorá vráti počet ovladateľných častí robota[7] Posun časti robota je realizovaný generovaním reálneho náhodného čísla pomocou metódy `RaveRandomFloat()` zmenšeného o konštatnú hodnotu $0,5$.

Náhodná simulácia sa zahajuje stlačením tlačidla *start.* Po jej spustení je automaticky deaktivovaný `groupBox`, v ktorom sa nachádzajú prvky na ovládanie robota a inštančná premenná `simulate` je nastavená na hodnotu *true.* Táto premenná riadi beh simulácie. Po stlačení tlačidla *stop* je táto premenná nastavená na hodnotu *false,* čím je ukončený beh simulácie. Súčasne sú aktivované tlačidlá na ovládanie pohybov robota. Pokiaľ je zvolený pohľad z kamery č. 5 alebo 6, zaostrí sa na aktuálnu pozíciu kamery pomocou metódy `focusCam()`.

Priebeh náhodnej simulácie:

```
pokiaľ (simulate == true) {
  náhodné_číslo = RaveRandomInt();
  robot[náhodné_číslo] += RaveRandomFloat();
  kontrola kolízií;
  prekreslenie scény;
  ak (vybratá kamera5 alebo vybratá kamera6) {
    zaostri kameru;
  }
}
```

[6] Viacnásobným pohybom sa rozumie dlhšie držanie tlačidla pre ovládanie pohybu
[7] DOF = Degrees of Freedom, uvedené v súbore *openrave.h.*

5.5 Ovládanie pohybov

Ovládanie pohybov je najpodstatnejšia časť celej aplikácie, preto je na ňu kladený veľký dôraz pri výbere vhodnej alternatívy na jej realizáciu. Implementácia ovládania pohybov by bola možná viacerými spôsobmi. Napríklad priamym zadávaním číselnej hodnoty, o ktorú má daná časť byť otočená. Do úvahy pripadá aj možnosť, kde by boli iba dve tlačidlá s možnosťami + a - a musela by sa explicitne zvoliť práve ovládaná časť užívateľom. Rozhodol som sa však pre alternatívu obsahujúcu jeden obrázok a dve prislúchajú tlačidlá, ktorá je relatívne jednoduchá na implementáciu a ľahko pochopiteľná nezaujatým užívateľom. Práca naviac je však extraktovanie bitmáp, ktoré reprezentujú jednotlivé časti robota.

Celkovo je možné v aplikácii pohybovať šiestimi časťami robota. Všetky pohyby robota reguluje užívateľ tlačidlami označenými + a - umiestnenými vedľa bitmapových obrázkov, ktoré reprezentujú jednotlivé ovládateľné časti robota. Po stlačení jedného z tlačidiel sa zavolá metóda `MainWindow::move(int num, bool plus)`. Parameter *num* je identifikátor (celé číslo z intervalu $< 6, 11 >$) konkrétnej časti robota. Parameter *plus* nadobúda hodnotu *true,* pokiaľ bolo stlačené tlačidlo *+*. V opačnom prípade nadobúda hodnotu *false.* Všetky pohyby sú vlastne otočením okolo svojej osi o konštantnú hodnotu $0,1$ v kĺbe, ktorý spája konkrétne dve časti. Pohyby sú limitované hodnotami uvedenými v definícii modelu medzi značkami `<limitsdeg>...</limitsdeg>`. Pokiaľ je zvolený pohľad z kamery č. 5 alebo 6, zaostrí sa na aktuálnu pozíciu kamery pomocou metódy `focusCam()`.

Priebeh funkcie *MainWindow::move(int num, bool plus):*

```
ak (plus == true) {
  robot[num] += 0.1;
}
inak {
  robot[num] -= 0.1;
}
ak (vybratá kamera5 alebo vybratá kamera6) {
  zaostri kameru;
}
```

5.6 Lokalizácia

Ponúkajú sa dve možnosti ako zabezpečiť dostupnosť aplikácie vo viacerých jazykoch:

- Načítať lokalizáciu pri spustení aplikácie, bez možnosti jej dynamickej zmeny. Pri tomto postupe je nutné mať vytvorený súbor, v ktorom bude údaj o tom, ktorý jazyk sa má načítať. Pri zmene jazyka je nutný reštart aplikácie,
- Umožniť zmenu jazyka aj za behu aplikácie. Nie je potrebné načítavať informáciu o tom, aký jazyk použiť. Treba však mať nastavený predvolený jazyk, v ktorom sa aplikácia spúšťa.

Platforma Qt4 umožňuje jednoduchú dynamickú zmenu jazyka využitím triedy `QTranslator`. Preto je vhodné využiť tieto výhody aj v našej aplikácie. Na tvorbu prekladov sa dá využiť automatizovaný nástroj *Qt Linguist* distribuovaný spoločne s knižnicami Qt.

Postup vytvorenia súborov s prekladom:

1. Pridanie riadku `TRANSLATIONS=rave_sk.ts`. Súbor *ts* je *XML* súbor, ktorý obsahuje preklady textových reťazcov,
2. Spustenie `qmake`,
3. Spustenie nástroja `lupdate rave_sk.ts`. Načítajú sa všetky preložiteľné textové reťazce zo zdrojového kódu do *.ts* súboru,
4. Otvorenie súboru *.ts* nástrojom *Qt Linguist.* V tejto aplikácii sa zobrazia nepreložené textové reťazce, ku ktorým je potrebné dopísať ekvivalent v požadovanom jazyku,
5. Spustenie nástroja `lupdate rave_sk.ts` Tento nástroj prevedie *.ts* súbor na binárny súbor *rave_sk.qm,* ktorý využíva trieda *QTranslator* (Načíta sa volaním metódy `QTranslator::load("file")`).

V zdrojovom kóde je potrebné preimplementovať virtuálnu metódu `QWidget::changeEvent(QEvent *event)`, v ktorej reagujeme na zmenu jazyka.

5.7 Definovanie modelu

Model simulovaného robota je definovaný v textovom formáte *XML*. Aplikácia očakáva súbor *model.xml,* ktorý je umiestnený adresári *data*. Ten sa nachádza v rovnakom adresári ako binárny súbor spúšťajúci celú aplikáciu. Štandardne je nadefinovaný model robotického ramena *Mitsubishi Melfa,* ktorý je umiestnený v areáli Fakulty informačních technologií v Brne. Úpravou tohto súboru je možné si nadefinovať ľubovoľný vlastný model. Treba však počítať s tým, že nebude zaručená plná kompatibilita s aplikáciou.

5.7.1 Komponenty robota

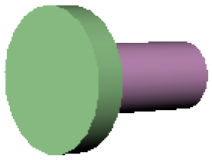
- **Telo robota** – telo robota sa definuje medzi značkami `<KinBody>...</KinBody>`. Atribútom `name="melfa"` definujeme názov modelu.
- **Súčasti** – jednotlivé časti tela modelu sa definujú medzi značkami `<Body>...</Body>`. Atribútom `name="názov"` sa dajú jednotlivé časti pomenovať. Atribút `type="[static | dynamic]"` určuje, či sa dá jednotlivou časťou pohybovať (hodnota *dynamic*) alebo sa jedná o nepohyblivú časť (hodnota *static*).
- **Kĺby** – kĺby (spoje) sa definujú medzi značkami `<Joint>...</Joint>`. Potom sa medzi značkami `<Body>...</Body>` uvedú komponenty, ktoré má daný kĺb spájať.
- **Senzory** – je možné definovať aj rôzne druhy senzorov. Pri implementácii boli použité senzory typu *BaseCamera,* ktoré reprezentujú kamery umiestnené na ramene robota. Senzory sú definované medzi značkami `<AttachedSensor>...</AttachedSensor>`, ktoré obsahujú značky `<sensor>...</sensor>`. Naviazanie na konkrétnu komponentu robota je definované medzi značkami `<link>...</link>`
- **Manipulátor** – medzi značkami `<Manipulator>...<Manipulator>` sú definované informácie o kinematickej postupnosti kĺbov. Je tu uvedený začiatok `<base>Base</base>` a koniec `<effector>Piece6</effector>`. Tieto informácie sú dôležité pri výpočtoch inverznej kinematiky.

Zjednodušená štruktúra *XML* súboru:

```
<Robot>
        <KinBody>
                <Body>...</Body>
                ...
                <Joint>...</Joint>
                ...
        </KinBody>
        <AttachedSensor>
                <sensor>...</sensor>
        </AttachedSensor>
        <Manipulator>...</Manipulator>
</Robot>
```

5.7.2 Príklad

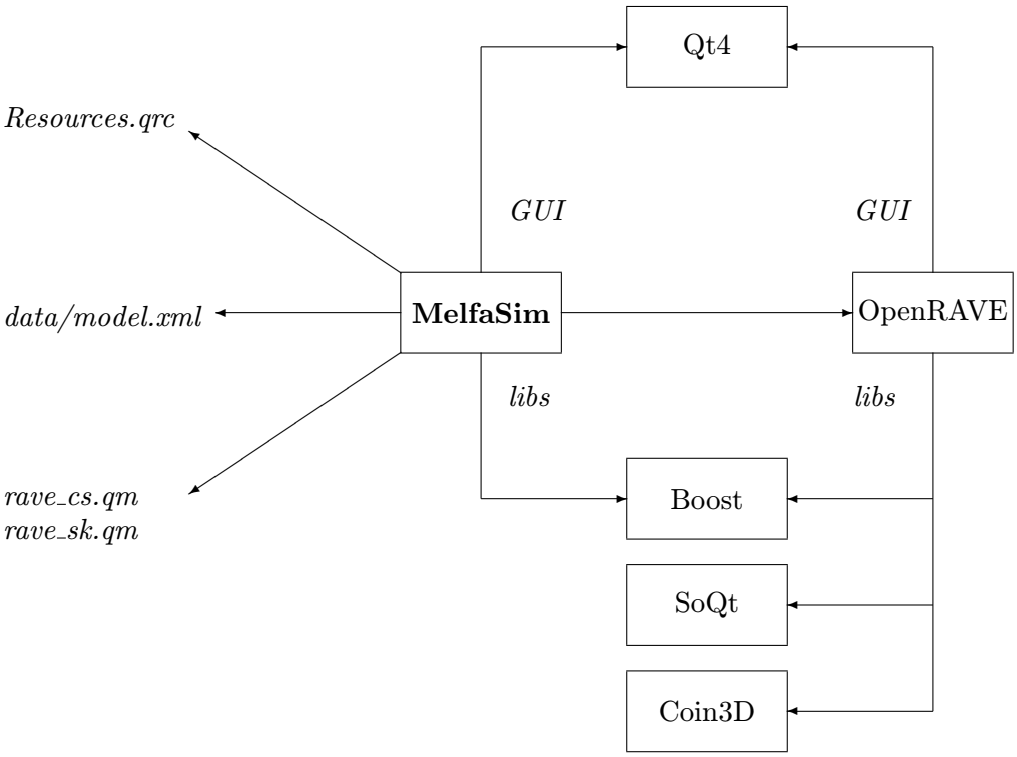
Na obrázku 5.3 je vidno spojenie dvoch komponent kĺbom. Príslušný *XML* kód aj so senzorom viď A.



Obr. 5.3: Spojenie dvoch častí kĺbom

5.8 Vnútorná štruktúra aplikácie

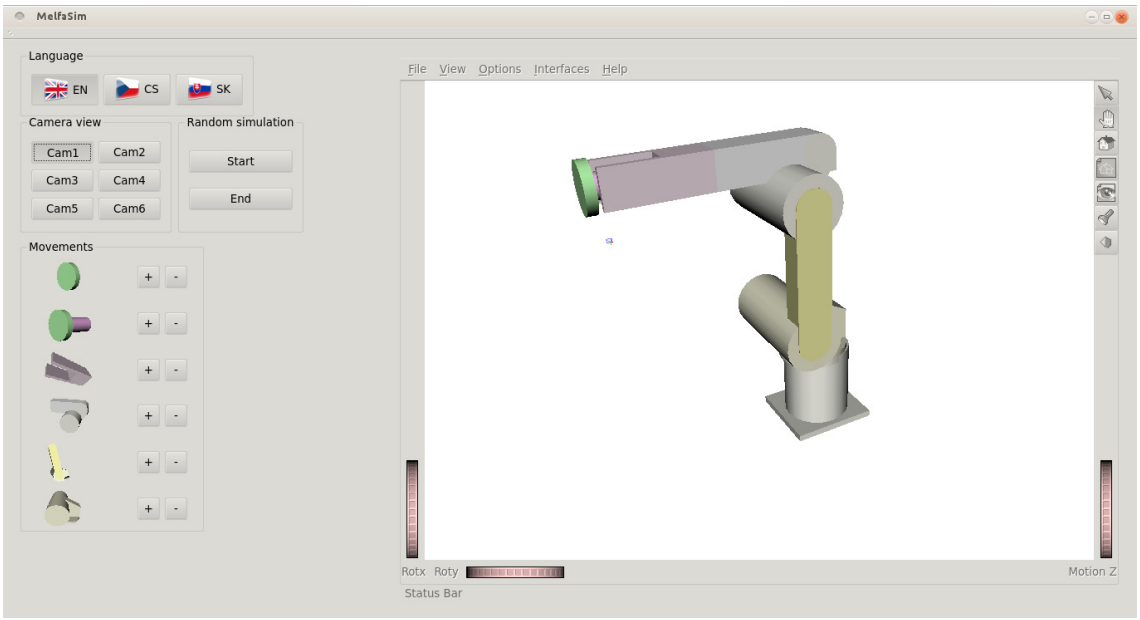
Na obrázku 5.4 je znázornená schéma potrebných knižníc a súborov pre chod aplikácie. Samotná aplikácia načítava súbory `Resources.qrc` (zdroje obrázkov, viď sekcia 5.1.2), `data/robot.xml` (definovaný model robota, viď sekcia 5.7) a `rave_cs.qm, rave_sk.qm.` (súbory s prekladmi aplikácie, viď sekcia 5.6) Aplikácia ďalej využíva knižnice *OpenRAVE, Boost* a *Qt4*. Samotný OpenRAVE ešte naviac *SoQt* a *Coin3D*. Nutnou súčasťou OpenRAVE sú aj pluginy, napr. *collada-parser* alebo *xmlreaders-core.*



Obr. 5.4: Štruktúra aplikácie

5.9 Snímok aplikácie

Na obrázku 5.5 je zachytená ukážka aplikácie v operačnom systéme *Ubuntu 12.10*. Na obrázku je zvolený predvolený pohľad na scénu z kamery č. 1 a nastavený predvolený jazyk angličtina. S takýmto nastavením privíta aplikácia užívateľa pri každom spustení. Sú ponechané ovládacie prvky grafického motívu daného operačného systému (možnosť *minimalizovať*, *maximalizovať* a *zavrieť*).



Obr. 5.5: Ukážka výslednej aplikácie

5.10 Problémy pri implementácii

Pri implementácii bolo nutné sa vyporiadať s viacerými problémami, ktoré skomplikovali vývoj aplikácie. Tu sú uvedené niektoré z nich:

- Nekompatibilita knižnice Qt5 s OpenRAVE. Nutnosť použitia staršej verzie Qt4 (konkrétne Qt 4.8.3),
- Nefunkčnosť senzorov vo verzii OpenRAVE 0.8.0. Nutná aktualizácia na verziu 0.9.0,
- Náročná inštalácia OpenRAVE na linuxových distribúciach. Najmä vo *fedora-based* distribúciach je slabá podpora závislých balíčkov,
- Pri OS Windows nebolo možné nainštalovať knižnice *Boost* v požadovanej verzii 1.44, na ktorých je závislý OpenRAVE. Nutnosť manuálnej kompilácie knižníc *Boost* zo zdrojových kódov,
- Pri OS Windows nekompatibilita závislých *.lib* knižníc s kompilátorom *MinGW*. Nutnosť kompilovať zdrojové kódy pomocou kompilátora *MVSC2010*[8],
- Pri OS Windows blokovanie aplikácie zo strany antivírusového programu.

[8] Microsoft Visual Studio C++ Compiler

Kapitola 6

Testovanie

Pri tvorbe aplikácií je treba klásť veľký dôraz na testovanie vybranými užívateľmi. Výsledky nám napovedia, ako ďalej rozvíjať našu aplikáciu, resp. na čo si dať nabudúce pozor. Testovať užívateľov je možné viacerými spôsobmi. V tomto prípade som sa rozhodol pre osobnú prezentáciu a komunikáciu s okamžitou spätnou väzbou. Testovanie aplikácie prebiehalo jednak vyskúšaním jej funkčnosti na rôznych sytémoch, ale aj praktickým testom užívateľmi.

6.1 Testy na rôznych OS

Aplikácia bola testovaná na rôznych operačných systémoch (konkrétne Linux a MS Windows) v 32-bitovej aj 64-bitovej verzii. Na každej testovanej platforme (okrem MS Windows XP[1]) bola aplikácia skompilovaná a spustená. Vo verzii *OpenRAVE 0.8.0* dochádzalo k problémom pri využití senzorov (pohľad z kamier č. 5 a 6). Taktiež neboli detekované niektoré kolízne situácie. Tieto implementačné problémy boli odstránené vo verzii *OpenRAVE 0.9.0*. V prípade spúšťania aplikácie na OS Windows pri aktívnom antivírusovom programe bolo nutné aplikáciu pridať do výnimiek alebo dočasne deaktivovať antivírus, resp. jeho rezidentný štít. Výsledky testovania a testované systémy sú uvedené v tabuľke 6.2.

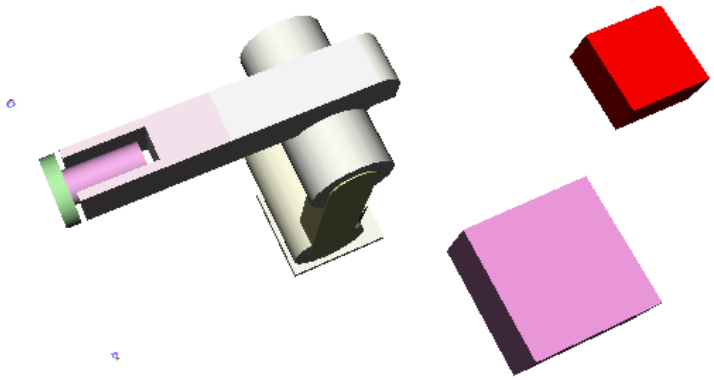
Operačný systém	**Jadro**	**OpenRAVE**	**Výsledok**	**Poznámka**
Ubuntu 12.10 32-bit	3.5	0.8.0	funkčné	nefunkčné senzory
Ubuntu 12.10 32-bit	3.5	0.9.0	funkčné	
Linux Mint 14 64-bit	3.5	0.9.0	funkčné	
Fuduntu 2013.2 32-bit	3.8.8	0.9.0	funkčné	
MS Windows 7 64-bit	-	0.9.0	funkčné	
MS Windows XP 32-bit	-	0.9.0	funkčné	

Tabuľka 6.1: Výsledok testov na rôznych OS

[1]Na MS Windows XP bola použitá verzia skompilovaná na MS Windows 7

6.2 Užívateľské testy

Aplikácia bola testovaná desiatimi užívateľmi, ktorí sa nezaoberajú problematikou robotiky ani simulácie. Hodnotená bola rýchlosť orientácie užívateľa v aplikácii a taktiež celkový dojem z aplikácie. Na obrázku 6.1 je znázornená aplikácia s testovacím modelom. Testovací model bol vytvorený upravením súboru *model.xml*, do ktorého boli pridané dva objekty. Úlohou užívateľov bolo čo najrýchlejšie sa dotknúť robotickým ramenom červeného a fialového objektu. Pri teste bol užívateľ nútený prepínať pohľad z kamier, pretože napr. fialový objekt bol viditeľný iba z kamery č. 3.



Obr. 6.1: Testovací model

Výsledky užívateľských testov sú uvedené v tabuľke 6.2. Všetky testy prebiehali osobnou konzultáciou a meraním času, ktorý užívatelia potrebovali na zorientovanie sa v aplikácii. V tabuľke je taktiež uvedený krátky komentár k aplikácii. Test každého užívateľa prebiehal nezávisle od predchádzajúceho, aby sa zabránilo prípadnému zvýhodneniu a aby boli testy čo najpresvedčivejšie. Užívatelia boli vybraní v rôznych vekových kategóriach s rôznym stupňom vzdelania. Bolo možné ohodnotiť celkový dojem z aplikácie známkami v rozsahu 1 až 5. Graf výsledkov známkovania je znázornený na obrázku 6.2.

Číslo	Vek užívateľa	Čas [min]	Komentár	Známka
1	19	1:40	OK	1
2	12	1:58	Dobrá hračka	2
3	19	2:10	OK	2
4	26	2:30	Nepraktické, k ničomu	4
5	22	1:25	OK	1
6	16	3:17	Na čo to je?	3
7	18	1:30	Nevidím v tom zmysel	2
8	31	2:45	OK	1
9	48	3:00	OK	2
10	46	4:40	Nebaví ma to	3

Tabuľka 6.2: Výsledky užívateľských testov

Obr. 6.2: Známkovanie aplikácie

6.3 Nedostatky a možné rozšírenia aplikácie

Na základe užívateľských testov boli zistené niektoré nedostatky aplikácie a bolo navrhnutých niekoľko možných rozšírení:

- Možnosť pomocou aplikácie ovládať skutočné robotické rameno,
- Lepšie označenie kamier. Pri označení číslami užívateľ nevie presne identifikovať, aký pohľad prislúcha k danej kamere,
- Spustenie aplikácie v jazyku operačného systému,
- Väčšia interaktivita pri ovládaní robota, možnosť ovládať pohyby klikaním priamo na robota,
- Väčšia plocha na zobrazenie simulačného modelu.

Kapitola 7

Záver

V tejto práci som sa zoznámil so simulačným nástrojom OpenRAVE, pomocou ktorého bola vytvorená jednoduchá aplikácia na simuláciu pohybov skutočného robotického ramena. Práca bola zameraná jednak na simuláciu, ale neodmysliteľnou súčasťou bolo aj vytvorenie intuitívneho grafického užívateľského rozhrania. Pri práci som sa naučil vytvárať komplexné grafické aplikácie pomocou nástroja Qt v operačných systémoch Linux a Windows. Taktiež som sa naučil pracovať so simulačným nástrojom OpenRAVE s definovaním modelu pomocou *XML* súboru a využívať závislosti potrebných knižníc pri tvorbe aplikácie.

Pri návrhu aplikácie bolo potrebné zohľadniť všetky možnosti a schopnosti skutočného robota, aby jeho model čo najviac odpovedal skutočnosti. To isté platí aj o pozícii kamier, ktoré snímajú robotické rameno z rôznych uhlov. Pri implementácii som sa snažil využiť všetky dostupné prostriedky nástroja OpenRAVE, aby bola simulácia pokiaľ možno čo najreálnejšia. Pri tvorbe užívateľského rozhrania bol kladený dôraz na jednoduchosť a praktickosť. Pri pozorovaní užívateľov pri vykonávaní testu aplikácie pomerne rýchlo pochopili funkcionalitu aplikácie. Na základe výsledkov testov môžem tvrdiť, že aplikácie je dostatočne intuitívna a funkčná.

Aplikácia nie je určená širokej verejnosti, svoje uplatnenie nájde pravdepodobne len u užívateľov, ktorí dochádzajú ku styku s podobnými nástrojmi alebo vlastnia konkrétny model robota. V tom prípade môžu využiť túto aplikáciu na jeho prezentovanie. Súčasťou práce bolo aj zhodnotenie aplikácie vybranými užívateľmi. Niektorých práca zaujala, iní zase nevidia rozumný spôsob využitia tejto aplikácie v ich prospech. Na základe výsledkov testov boli zistené prípadné nedostatky a možné vylepšenia aplikácie.

V budúcnosti by som sa rád zaoberal nedostatkami aplikácie a vytvoril novú verziu, v ktorej by som čo najviac vyšiel v ústrety užívateľom. Takisto by som rád vytvoril rozhranie medzi simuláciou a konkrétnym robotom, ktoré by umožňovalo aj jeho reálne ovládanie.

Literatúra

[1] *Computer simulation*, [online]. 2007 [cit. 2013-02-11].
Dostupné z:
http://www.sciencedaily.com/articles/c/computer_simulation.htm

[2] *VirtualRobot Simulator*, [online]. 2010 [cit. 2013-02-11].
Dostupné z: http://robotica.isa.upv.es/virtualrobot/

[3] *Circuit Simulator*, [online]. 2012 [cit. 2013-05-06].
Dostupné z: http://www.falstad.com/circuit/directions.html

[4] *Gazebo*, [online]. [cit. 2013-04-10].
Dostupné z: http://gazebosim.org/about.html

[5] Diankov, R.: *Automated Construction of Robotic Manipulation Programs*. Dizertačná práca, Carnegie Mellon University, Robotics Institute, August 2010.
Dostupné z: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[6] Jasmin Blanchette, M. S.: *C++ GUI Programming with Qt 4*. Trolltech ASA, 2008, ISBN 0-13-235416-0.

[7] Karel Mozdřeň: *Programování v C++ První kroky*. [online]. 2009 [cit. 2013-02-19].
Dostupné z: http://homel.vsb.cz/~moz017/cpp/kniha/c++.pdf

[8] Noskievič, P.: *Simulace systémů*. Ostrava, Česká republika: Vysoká škola báňská, 1992, ISBN 80-7078-112-2.

[9] Straka, M.: *Simulácia diskrétnych systémov a simulačné jazyky*. Košice, Slovenská republika: Edičné stredisko / AMS, Fakulta BERG Technickej univerzity v Košiciach, 2005, ISBN 80-8073-289-2.

[10] Záda, V.: *Diferenciální rovnice dynamiky robotů a jejich základní vlastnosti*. Liberec, Česká republika: Technická univerzita v Liberci, 2010.

Príloha A

XML model

```
<Body name="Piece5" type="dynamic">
        <mass>
                <density>.1</density>
        </mass>
        <Translation>0.0 0.0 0.0</Translation>
        <Geom type="cylinder">
                <Translation>-0.499 0.83 0.0</Translation>
                <rotationaxis>0 0 1 90</rotationaxis>
                <diffuseColor>80 80 10</diffuseColor>
                <radius>.032</radius>
                <height>.138</height>
        </Geom>
</Body>
<Joint name="kloub5" type="hinge" circular="false">
        <body>Piece4</body>
        <body>Piece5</body>
        <weight>1</weight>
        <limitsdeg>-120 120</limitsdeg>
        <axis>0 0 1</axis>
        <anchor>-0.515 0.83 0.0</anchor>
</Joint>
<Body name="Piece6" type="dynamic">
        <mass>
                <density>.1</density>
        </mass>
        <Translation>0.0 0.0 0.0</Translation>
        <Geom type="cylinder">
                <Translation>-0.5815 0.83 0.0</Translation>
                <rotationaxis>0 0 1 90</rotationaxis>
                <diffuseColor>80 10 80</diffuseColor>
                <radius>.064</radius>
                <height>.027</height>
        </Geom>
</Body>
```

```
<AttachedSensor name="cam5">
        <link>Piece5</link>
        <translation>-0.6 0.8 0.3</translation>
                <rotationaxis>1 0 0 180</rotationaxis>
        <sensor name="cam5" type="BaseCamera">
                <KK>640 640 320 240</KK>
                <width>640</width>
                <height>480</height>
                <framerate>5</framerate>
                <color>0.5 0.5 1</color>
        </sensor>
</AttachedSensor>
```

Príloha B

Obsah CD

`src/` – adresár so zdrojovými súbormi aplikácie
`bin/` – skompilovaný program
`tex/` – zdrojové súbory textu technickej správy
`manual.pdf` – užívateľská príručka
`melfasim.pdf` – technická správa
`README.txt` – súbor s popisom aplikácie
`INSTALL.txt` – súbor s návodom na inštaláciu aplikácie